(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 March 2001 (08.03.2001)

PCT

(10) International Publication Number
WO 01/16731 A1

(51) International Patent Classification[7]: G06F 9/445

(21) International Application Number: PCT/US00/22638

(22) International Filing Date: 17 August 2000 (17.08.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/385,779 30 August 1999 (30.08.1999) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 09/385,779 (CIP)
Filed on 30 August 1999 (30.08.1999)

(71) Applicant *(for all designated States except US)*: **NEOPLANET, INC.** [US/US]; 1424 West 14th Street, Tempe, AZ 85281 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **FRISKEL, James** [US/US]; 3221 S. Atlantic Avenue #704, Cocoa Beach, FL 32931 (US). **MARTINEAU, Carl, Preston** [US/US]; 2736 S. Yucca Street, Mesa, AZ 85202 (US). **COX, Bradley, George** [US/US]; 7348 W. Crest Lane, Glendale, AZ 85310 (US). **SIELAFF, Martin, Ernst** [US/US]; 1581 E. Erie, Chandler, AZ 85225 (US). **COHEN, Drew** [US/US]; 9215 N. 107th Way, Scottsdale, AZ 85258 (US).

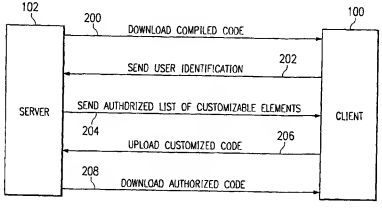
(74) Agents: **BRUCE, T., Neel** et al.; Streich Lang, P.A., Renaissance One, Two North Central Avenue, Phoenix, AZ 85004 (US).

(81) Designated States *(national)*: AE, AL, AM, AT, AT (utility model), AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, CZ (utility model), DE, DE (utility model), DK, DK (utility model), DM, EE, EE (utility model), ES, FI, FI (utility model), GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR (utility model), KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (utility model), SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European

[Continued on next page]

(54) Title: METHOD AND SYSTEM FOR MODIFYING COMPILED CODE USING A CUSTOMIZATION TOOL



(57) Abstract: A method and system for modifying complield code without access to the underlying source code using a customization editing tool residing on a client computer (100). The compiled code (200) is initially downloaded from a server computer (102) to the client computer (100). The customization tool contacts the server computer (102) to determine the changes to the compiled code that are authorized. A user on the client computer (100) runs the tool and is presented with authorized customization choices by the tool. After the user has made the desired modifications, the customized code (206) is uploaded to the server computer (102). If the customized code conforms to the initial authorization provided by the server computer (102), then the customized code is modified to embed an authorization code. The authorized code (208) is then sent to the client computer (100) for normal usage by the user. The user does not need access to the source code to implement the changes to the compiled code, which may include changes in functionality and the look and feel of the corresponding application program. The user also does not require access to the installation source code to implement custom installations on the user's computer. The installation program is pre-compiled, but configurable to mimic a custom-generated installation program.

WO 01/16731 A1

patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Published:

— *With international search report.*

METHOD AND SYSTEM FOR MODIFYING COMPILED CODE USING A CUSTOMIZATION TOOL

FIELD OF INVENTION

The present invention relates to modification of computer programs in general and, more particularly, to a method and system for modifying compiled code using a customization editing tool.

BACKGROUND OF INVENTION

Software application programs in general have been previously customized by making changes to the source code used to create one or more files compiled and linked together to create executable versions of the programs (also referred to herein as executable or compiled code). Previously, in order for a user of an application program to make changes to the executable code of the program, for example to customize its user interface or other functionality for marketing or other industry-specific purposes, the user necessarily needed access to the source code. The typical prior procedure was for the user to modify the source code as appropriate for the desired customization or functionality changes and then to re-compile the program from the modified source code. However, a disadvantage of this prior approach is that the user must be given access to the source code, which often contains trade secrets or other confidential information that a proprietor desires to maintain in confidence.

Shared libraries provide an alternative approach to the above for modifying an executable code version of an application program. A shared library may be linked with an executable program at run time instead of at compile time. Shared libraries are currently used on many types of platforms. One specific shared library used on a MICROSOFT WINDOWS platform is a dynamic link library (DLL). With dynamic linking (as for shared libraries in general), a user can create its own DLLs that are

called by the main executable program as necessary at run time in order to customize the application program. However, the creation of DLLs still requires a user to obtain significant information about the source code from an Application Program Interface (API) provided by the creator of the source code. Also, the creator of the executable program does not have control over the activities initiated by the DLL. Specifically, the executable program source code defines one or more functions that are called in the DLLs. However, the user can make those functions do almost anything desired by the user. This is a disadvantage because the actions initiated by the user's DLL may conflict with other functionality or data structures of the calling executable program. In addition, troubleshooting of the main executable program becomes more difficult with DLLs because the developer of the executable program cannot controllably define the range of actions that may be requested by the user in a DLL. Further, the creation of DLLs based on the API requires computer programming skill so that lay persons are not able to readily customize a program by creating DLLs. Among the things that a user would need to successfully create a DLL include the right version and type of compiler for compiling the DLL and the correct type of platform corresponding to the operating system and central processing unit.

One specific application program for which it is desirable to customize the program for specific users is an Internet or web browser, which may be used for example to access and navigate the Internet or a private intranet. Currently-available Internet browsers, such as INTERNET EXPLORER from Microsoft Corporation and NETSCAPE NAVIGATOR from Netscape Communications, Inc., do not permit a user to modify the executable code to customize the browser. Thus, a user cannot readily customize these browsers to make changes that may be of advantage in a specific application area or industry. Further, an entity having authority to distribute additional copies of such a browser may desire, but be unable, to readily make changes

to the browser interface to create a version with the entity's brand name or other features proprietary to the entity.

Accordingly, a need exists for a tool that permits a user, even one without software programming skill, to make substantial changes to the functionality of an application program by modifying its executable code without having access to the underlying source code and without going through a recompilation process.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a computer system comprising a server computer and a client computer running an editing tool for customizing compiled code according to the present invention;

FIG. 2 illustrates an overview of client-server data communications for the system of FIG. 1;

FIG. 3 is a flow diagram representing a preferred process for generating customized code using the editing tool of FIG. 1;

FIG. 4 is a flow diagram representing a preferred process for modifying the compiled code as part of the process of FIG. 3;

FIG. 5 is a flow diagram representing a preferred process for authorizing the compiled code received by the server computer or acceptance unit of FIG. 1;

FIG. 6 is a flow diagram representing a preferred process for the client computer to determine whether the compiled code has been altered after its authorization;

FIG. 7 illustrates an example of a customizable element in source code form before and after customization by the client computer; and

FIG. 8 illustrates an example of an unlock variable in source code form before and after authorization of the compiled code by the server computer.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The detailed description which follows is presented largely in terms of procedural operations to be performed with respect to computer files within one or more computer memories. Such operations include, among other things, generating, compiling, modifying, comparing, linking, and storing computer source code and compiled code. These procedural descriptions and representations are the approaches used by those skilled in the software and data processing arts to most effectively convey the substance of their work to others skilled in the art.

The above operations require physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical, optical, or magnetic signals capable of being stored, transferred, combined, compared, or otherwise manipulated. It is convenient at times, principally for reasons of common usage, to refer to these signals as values, elements, characters, numbers, or the like. It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities.

The present invention provides a method and a system that permits a user, even one without significant software programming skill, to make substantial changes to the functionality of an application program, for example an Internet browser, by modifying its executable code without having access to the underlying source code and without going through a recompilation process. A significant advantage of the present invention is that third parties may make substantial changes to the appearance and functionality of an application program by directly modifying the underlying compiled executable code in this way.

In contrast to prior approaches using shared libraries such as DLLs, the present invention provides the advantage of controlling the range of modifications that a user may make to the executable code of the program. Thus, the quality of the software is

less likely to be compromised by customizations done by the user and the reliability of the browser program (also referred to herein as simply a "browser") operation will be improved.

The modifications of the present invention are made directly to the main executable code so that the browser opens and runs immediately in its default state as customized by the user. This is in contrast to the prior approach where a user may do some customization of a browser after it has been installed in its default state. For example, third parties may use the present invention to create their own customized version of the browser, which may include their own brand name and appearance. The modifications are readily made by the user's simply responding to a list of authorized changes that are presented to the user by an editing tool provided to a user with a copy of the browser executable code in its default, distributed state. By answering the questions and instructing the editing tool to modify the executable code, the user can modify the browser executable code without access to the source code. Instead, the information necessary to make the modifications is embedded within the editing tool. The distribution of a customized browser provides a user such as a software distributor with a convenient way to build customer recognition of the user/distributor's brand, customer loyalty, and the size of customer installed base. Internet browser programs that may be used to advantage in conjunction with and modified by the system and method of the present invention are generally described in pending U.S. Application No. 09/359,436 filed July 22, 1999, by James Friskel and Michael Noe, titled NETWORK BROWSER METHOD AND SYSTEM FOR CONTROLLING BROWSER OPERATION, CUSTOMIZING BROWSER USER INTERFACE, AND PROVIDING UPDATABLE CHANNELS FOR THE SELECTION OF NETWORK FILES, which is hereby incorporated herein by reference.

One specific example of an Internet browser that may be customized by a user is the browser distributed by NeoPlanet, Inc., of Phoenix, Arizona, on its Internet web

site at www.neoplanet.com, which browser and related web site information is hereby incorporated herein by reference. The NeoPlanet browser permits the customization of its user interface by enabling a user to select and change the appearance of the browser by the selection of skins, which are files containing graphical and other information used to create a particular look and feel for the browser, from libraries created and provided by NeoPlanet or third parties or the user itself. Also, the NeoPlanet browser permits a user to customize lists of web pages presented to the user in an interface window for the user's selection of new web pages for viewing and browsing. These lists are organized as a multiple level hierarchy with the top level organized into categories referred to as channels and lower levels organized underneath these channels (also referred to herein as a "channel hierarchy"). The user is able to customize this channel hierarchy by selecting files of channel information created and provided by NeoPlanet or a third party.

By the use of the present invention, a user may customize the NeoPlanet browser, and also other browsers, to have a look and feel and functionality that is specific to the user's desired requirements and, according to the present invention, these customizations may be made directly to the browser executable code so that the default state of the browser, for example when copied and further distributed to third parties by the user, will be the state as customized by the user and without the need for any further action by subsequent parties to which the software has been distributed.

The present invention relates to an apparatus and method for operating a computer system, such as computer system 103 of FIG. 1, and processing electrical or other physical signals to generate other desired physical signals. The apparatus of the present invention may be specially constructed for the required purposes, or it may comprise general purpose computers in a general communications system, such as in computer system 103 of FIG. 1, selectively configured by computer programs stored within the memories of the client and server computers and the acceptance unit. The

generally required structure for a variety of computer systems that may be constructed will be apparent to those of skill in the art from the description given below.

The present invention also relates to apparatus comprising a set of computer programs stored on electrical, magnetic, or other physical media, and designed so as to control and instruct general purpose computers such as the client and server computers and acceptance unit of FIG. 1 to perform the operations described herein. The general structure required for such computer programs will also appear from the description given below, and those of ordinary skill in the art of computer programming may implement various embodiments of such programs using a wide variety of programming languages.

The above features and other inventive features of the present invention are described below with reference to specific embodiments as illustrated in the figures listed above. Useful machines for performing the operations of the present invention include general purpose digital computers or similar devices. FIG. 1 illustrates general purpose client and server computers suitable for implementing these operations and in electronic communication with each other. With specific reference to FIG. 1, computer system 103 comprises a server computer, an acceptance unit, and a client computer running an editing tool for customizing compiled code according to the present invention. It should be noted that "compiled code" is often known as "object code", but the term "compiled code" is used herein. As described herein, computer system 103 is generally described with reference to Internet-based communication between the client and server computers and acceptance unit, but the disclosed computer system may be employed in other on-line and remote computer systems, such as on-line services using proprietary user interfaces as well as private intranets and extranets.

More specifically, as shown in FIG. 1, client computer 100 (also referred to as a "client"), for example a personal computer (PC) running the WINDOWS operating

system from Microsoft Corporation or a UNIX-based computer, is in two-way electronic communication 101 with server computer 102 (also referred to as "server"). Communication 101 is preferably done using Internet protocol-based communication. Client 100 comprises monitor 104 for visual monitoring by a user, central processing unit (CPU) 106 for running software programs on client 100, and storage device 108 for storing software programs in a permanent form on magnetic, optical, or other storage media, such as for example a hard disk drive. Customization editing tool 110 is a software program according to the present invention, stored in one or more associated files, for customizing compiled code on client 100. Tool 110 implements the methods of the present invention as described herein. Tool 110 resides on storage device 108 and runs on client 100 using CPU 106 and other related conventional computer components. CPU 106 is preferably a standard, commercial digital microprocessor, such as an Intel PENTIUM processor. Tool 110 is preferably implemented in software using appropriate techniques suitable to implement the procedures described herein in one of many standard programming languages such as, for example, MICROSOFT VISUAL C++.

Server 102 comprises monitor 112, CPU 114, and storage device 116 on which authorization program 120 is stored. Storage device 116 may be of a similar type of device as discussed above for storage device 108. Authorization program 120 is organized in one or more associated files, runs on server 102 using CPU 114 and other related conventional computer components, and may be implemented in software using conventional techniques as described above for tool 110.

Acceptance unit 122 is a general purpose computer in two-way electronic communication 121 with client 100. Communication 121 is initiated by tool 110 as appropriate for sending compiled code that the user of client 100 has customized to the user's satisfaction for authorization for normal usage, for example distribution to the public, as discussed further below. As also discussed below, this authorization is a

final control step that is preferably performed by the operator of server 102 to: (i) confirm compliance of the customized compiled code with operator-desired specifications or terms of any agreement, for example regarding user branding using the user's trademarks, previously reached between the operator and the user; and (ii) ensure that no obscene or other objectionable material has been included in the compiled code during customization. Communication 121 is preferably Internet protocol-based communication as for communication 101. In one approach, communication 121 may be the sending of an electronic mail (also known as E-mail) message from client 100 as initiated by tool 110 and that includes the customized code as an attachment, and the reply from acceptance unit 122 to client 100 may be another E-mail having a copy of an authorized version of the customized code as an attachment. Also, other conventional methods of electronic or Internet communications can be used to transfer the customized code. It should be noted, however, that in other embodiments of the present invention acceptance unit 122 is not necessary and all communications with client 100, including communications regarding authorization of customized compiled code, could be accomplished through server 102. Also, acceptance unit 122 stores a copy of one or more portions of authorization program 120 as appropriate for those operations that it may perform.

Basic System and Methodology

An overview of the preferred approach for customizing compiled code in accordance with the present invention is illustrated by FIG. 2, which illustrates client-server and acceptance unit communication data flows for the system of FIG. 1 during implementation of the operations described herein. A copy of compiled code that has been previously created from the source code corresponding to an application program, such as an Internet browser, is stored on server 102. The compiled code is generated using standard compiling and linking procedures. The form of the compiled

code is the default form as set by the operator of server 102. The compiled code is preferably compressed using standard techniques and downloaded at step 200 to client 100, for example in a self-extracting zip file. The compiled code may be sent, for example, by E-mail or the user may download it by accessing a webpage connected to server 102.

The user of client 100 stores the compiled code in storage device 108 and installs the compiled code using, for example, a setup program compiled using a WISE compiler available from Wise Solutions. The setup program extracts and installs the components of the compiled code into predetermined or a user-selected file directory on client 100. The downloaded compiled code includes the executable code for an application program, for example an Internet browser (described below as an example of a specific embodiment of the present invention), and editing tool 110. Tool 110 comprises components for customizing the executable browser code, providing a graphical user interface to the user of client 100, and a zipping program for re-compressing the customized compiled code for its return to server 102 or acceptance unit 122. Other files provided as part of the downloaded compiled code include any default customization files called by the application program during run-time, for example skin files and channel hierarchy files, and conventional software for installing and testing the application program on client 100.

When run by the user, tool 110 will preferably prompt the user for the user's E-mail address and a password previously provided by the operator of server 102. The password is preferably a unique user identification and at step 202 is sent by tool 110 to server 102. The password is processed by authorization program 120 to determine the extent to which the user is authorized to modify the application program. Each password will correspond to one of several varying levels of authorization. Alternatively, if the user does not have a password, then the user will only be

authorized to make modifications as the lowest authorization level as pre-defined by authorization program 120.

The portions of the compiled code that the tool is able to modify are referred to herein as customizable elements. Such elements include items relating to the functionality and the appearance of the application program to be customized on client 100. For example, such elements include text labels and graphic images presented during start-up, on pull-down menus, on help and other pop-up windows, and on other portions of the user interface presented to the user during operation. Further examples include items that can be enabled or disabled such as certain scrolling banners, headlines in certain windows, and the presentation of functional options that are presented to the user. Moreover, the network location of certain files accessed by the browser program during start-up and operation such as libraries of skin files or channel hierarchy files as used, for example, to customize the NeoPlanet browser appearance and channel navigation features may be customized by providing new network file location designations or indicators such as, for example, uniform resource locators (URLs) or directory paths corresponding to local storage on client 100 or a local network. These file location designations may be customizable elements. Yet other features such as privacy features or functional configurations may be selected from predefined choices by the user and these choices may also be customizable elements.

At step 204, a list of those customizable elements that the user is authorized to modify is sent to client 100 by server 102. Tool 110 then presents only the authorized elements to the user preferably using a user-friendly graphical user interface (GUI). The user enters its customized choices and corresponding customized values on client 100, and when done, tool 110 saves these choices in a file stored locally on storage device 108. Also, tool 110 permits a user to recall prior customized choices and values from a saved file that may have been created previously. After all customization choices have been made and entered by the user, tool 110 is run to

actually perform the direct modification of the compiled code that executes the application program. This modification process is described in more detail below.

As noted above, server 102 is shown FIG. 2, but acceptance unit 122 can be optionally used for uploading customizing code at step 206 and downloading authorized code at step 208. Further, acceptance unit 122 and server 102 may optionally be in communication to coordinate handling and/or sharing of processing of the uploaded customized code.

At step 206, the final version of the customized code as modified by the user using tool 110 is compressed and sent to server 102, or optionally as described above to acceptance unit 122. After a determination as to whether the customized code meets desired requirements of the operator of server 102 and/or acceptance unit 122, the customized code is processed by another component of authorization program 120 used to create an authorized or unlocked version of the customized compiled code uploaded at step 206. Note that, as mentioned above, acceptance unit 122 stores a copy of one or more portions of authorization program 120 as appropriate for the operations it performs. It should be noted that prior to authorization, when run on client 100 or any other computer, the customized browser code will display an unauthorized usage message, such as in a pop-up window at start-up, stating that usage and distribution of the browser is not authorized.

At step 208, the authorized code is downloaded to client 100 for installation and running on client 100. The authorized code can then be further distributed by the user, and the unauthorized message mentioned above has been disabled so that it no longer appears.

Modification of Compiled Code

FIG. 3 is a flow diagram representing a preferred process for generating customized code on client 100 using customization editing tool 110. Note that in FIG.

3 and the following figures, rectangles represent steps to be performed, while ovals represent computer code or other information processed by, used in performing, or resulting from these steps. Oval 300 represents downloaded compiled code received by client 100 at step 200 of FIG. 2.

Initially, at step 301, the downloaded code, for example in the form of a self-extracting executable file as mentioned above, is installed by the user on client 100 to install editing tool 110, the executable compiled code for the default application program, and the other related files on client 100. The self-extracting file downloaded from server 102 may be created, for example, using the WISE compiler. As is standard, the WISE compiler will compile and compress several files into a single self-extracting file including an executable installation program to run the self-extraction process, the default browser executable program and its related DLL and data files, editing tool 110, and another customized browser installation program to be used by the user after modification of the browser executable program. It should be noted that the WISE compilation of the above files to form the self-extracting file is done on server 102 or another computer (not shown) associated with server 102, and part of this compilation uses an installation script source file to define the specifics of the process to be run by the compiled installation program on client 100.

As part of step 301, the downloaded self-extracting file is run to install the components mentioned above to either predefined or user-specified directories on client 100. During this installation process, the WISE installation program decompresses and installs each file from the self-extracting file to its appropriate location, for example, in a directory tree on storage device 108.

Also as part of step 301, editing tool 110 is run to initiate the customization process, which is done, for example, by the user's clicking on a desktop icon corresponding to editing tool 110. According to the preferred modification process executed by editing tool 110, at step 302 compiled code 300 is initially copied to a

new memory location on storage device 108 to provide copied code 304. Changes will be made to copied code 304 so that a default version of the original code is retained for possible further modification iterations. Copied code is modified at step 306, as controlled and limited by authorized customizable list 308 obtained from server 102 at step 204 of FIG. 2, to generate modified code 310. More specific details that are part of modification step 306 are discussed below with reference to FIG. 4.

Modified code 310 is executable code that has been directly modified by direct changes to one or more corresponding files on storage device 108. No access to source code or recompilation is necessary since the executable code is directly modified in an intelligent way as controlled by editing tool 110.

At step 312, modified code 310 is installed on client 100, or optionally another computer suitable for running modified code 310, using the customized browser installation program originally provided as part of the downloaded compiled code received by client 100 at step 200 of FIG.2. Specifically, this installation process preferably involves creating a self-extracting file on client 100 by compressing several files into a single file, for example, using the publicly-available zlib compression library available, for example, at URLs www.cdrom.com/pub/infozip/zlib/ or www.winimage.com/zLibDll/ or another standard compression program. The components of this self-extracting file will be the customized browser installation program mentioned above, the modified executable browser program and any modifications and/or additions to its associated DLL and/or data files, and an installation configuration file that is read by the customized browser installation program.

After creating the above self-extracting file, the installation process can be performed by the user on client 100 by executing the self-extracting file. Upon execution, the customized browser installation program will execute by installing the browser application program on client 100. The installation program operates

logically in an almost identical fashion to the WISE installation program except that the installation program installs the browser program and associated files from a predetermined source directory tree on storage device 108 rather than from within the same executable code as for the installation program (as is standard for a conventional WISE installation).

Editing tool 110, while executing modifications selected by the user, creates a configuration file that reflects the customizations made by the user in step 306. The configuration file is different from the manifest file discussed below and is used as a recipe to guide the installation process executed by the customized browser installation program above. The use of the configuration file permits the installation process to be dynamically customized by the user as described herein.

The customized browser installation program above can be created by writing and compiling the installation program using the standard WISE compiler. The changes required to write and compile the WISE program are readily apparent to one of skill in the art based on the operations described herein and the standard library functions provided with the WISE compiler. Alternatively, the installation process that is part of step 312 can be done using the INSTALL SHIELD product available from Install Shield.

It should be appreciated that one type of modification that can be made to the application program by editing tool 110 is to customize the final product's installation process itself. This customization is in addition to changes made to the functionality of the application program or to the DLL or data files it accesses during operation. For example, the installation program can be customized to make the customized browser the end-user customer's default browser. Also, the installation program can be customized to do a so-called "silent install" in which yet another setup program of a third party for a different application product can call the self-extracting file created

above and pass the customized browser installation program file destination information without the need to query the end-user customer during installation.

It is an advantage of the present invention that the user create a single self-extracting file as described above for installation of the customized browser program so that the user can fully check the operation of the program from an end-user customer's perspective. For example, an end-user customer will receive the browser program in the form of the single self-extracting file described above. By permitting the user to create this file itself on client 100 prior to sending the modified code 310 for authorization, the user can internally distribute and fully test the modifications and installation process as will be experienced by end-user customers. This significantly speeds the cycle of final product development and authorization for distribution as compared to prior approaches.

Also, an advantage of providing the user with the pre-compiled customized browser installation program above is that the program can use the logic and installation functions of the WISE compiler, and it is not necessary to provide the user with the WISE compiler itself. In the typical WISE compilation process, the executable installation program is compressed with all files to be later installed. However, this typical approach would not permit a user to make modifications to the browser program files on the client because the pre-compiled browser installation program would not function properly with the newly-modified files. Thus, the customized browser installation program uses the configuration file mentioned above. In an alternative embodiment, it should be noted that the customized browser installation program could be created using the INSTALL SHIELD program mentioned above. Also, yet other installation programs having the feature of accessing files from a source directory during installation could be used as the customized browser installation program.

After the above customized browser installation, modified code 310 is run by the user to determine if the user is satisfied with the installation process itself and the other customizations made. As mentioned above, modified code 310 will display a failed authorization message to the user when run. Otherwise, modified code 310 operates as it will in its final, authorized state (as discussed below). Thus, the user can fully test both the installation process and operation of modified code 310.

At step 314, if the user determines modified code 310 is not acceptable, then the process returns to step 302 and starts again with the creation of a new copy of downloaded compiled code 300 to provide a new version of copied code 304. Tool 110 queries the user for new changes to be made or reads a file of changes to be made that has been created by the user and is compatible with tool 110. At step 306, the modification process is repeated on the new version of copied code 304. Then, the install and run step 312 is repeated on the new version. This modification loop is repeated iteratively as necessary until the user accepts the changes to the code.

At step 314, if the user finds modified code 310 acceptable, then at step 316 the modified code 310 is compressed and zipped into a self-extracting file using conventional zipping and compression methods. For example, as similar to that mentioned above, this compression can be done using the publicly-available zlib compression library available, for example, at URLs www.cdrom.com/pub/infozip/zlib/ or www.winimage.com/zLibDll/. Preferably, however, the self-extracting file created as part of step 312 above during testing on client 100 is simply uploaded along with a manifest file (discussed below) created by editing tool 110. In general, the self-extracting file will be similar in type of content to the files originally downloaded to client 100 at step 200. The self-extracting file will contain modified code 310, other files for setting up and installing the modified code on another computer, and customization files that are called by the modified code during operation such as skins files or channel hierarchy files containing organized

and categorized URLs. The self-extracting file also is accompanied by an additional manifest file generated by editing tool 110. This manifest file is a list of all customizable elements associated with the downloaded compiled code and the customized values selected by the user and incorporated as modifications in modified code 310. It should be noted that the manifest file will only contain information regarding those customizable elements that the user was authorized to modify. Also, in the preferred embodiment, the manifest file is not used by client 100 and is only used during quality assurance testing by acceptance unit 122 or server 102, as may be applicable.

After the code is compressed at step 316, or if the same self-extracting file created at step 312 is being sent, at step 318 the customized code in the self-extracting file is uploaded to server 102 or preferably acceptance unit 122 as discussed above. This uploading is initiated and controlled through the user interface of tool 110 and corresponds to step 206 of FIG. 2. It should be noted that if the file is uploaded to server 102, in the preferred approach server 102 will still need to send the file to acceptance unit 122 or to another computer on which a quality assurance person can install and test the user-customized code. Server 102 itself is typically not directly used by a human operator, yet part of the preferred quality control process (as discussed below) includes human examination of the customized program's operation.

FIG. 4 is a flow diagram representing a preferred process for modifying the compiled code at step 306 in the process flow of FIG. 3. Copied code 304 is modified by searching for an embedded key that was placed in the original compiled code downloaded from server 102. The embedded key is associated with and embedded as part of the underlying source code for each customizable element in the compiled code. Preferably, the embedded key is a single, common key placed in many locations throughout the entire code, but in other embodiments more than one key could be used. The key is for example an embedded ASCII string as discussed further below.

At step 400, tool 110 searches in turn for each key in copied code 304 to locate each customizable element. For example, when the first key is located, tool 110 will have located the first customizable element 402, which has the default value that was set at compile time. The modification process involves a comparison of these default values with customized values that have been selected by the user. Authorized customizable list 308 is used at step 406 to create the user's customized values to generate list 408 of user customized changes. List 408 is preferably generated prior to searching for the first key at step 400. The manifest file mentioned above is a file, for example an ASCII file, containing list 408 and all associated values entered by the user.

At step 404, for each customizable element located by a key search, the default value of the key is compared to the user selected value in list 408 corresponding to that element. At step 410, if the user has changed the value from the default value, the new value selected by the user is entered at step 414 into copied code 304 by directly modifying the code, for example by changing an ASCII string embedded in the code. If the user has not changed the value, the default value is left unchanged at step 412. In either outcome, at step 416 the embedded key used to locate that customizable element is replaced with a customized flag indicator to indicate that the customizable element has been modified or the user has determined that the element is not to be changed. It should be noted that since the key has been replaced, the corresponding customizable element can no longer be modified by tool 110 since the key used to locate the element has been removed.

After step 416, the process continues by again searching for the next key at step 400. Each key in copied code 304 will be searched for and the corresponding customizable element modified or left unchanged. This iterative process continues until all keys have been searched and replaced with customized flags.

FIG. 7 illustrates an example of a customizable element in source code form before and after customization by the client computer. It should be appreciated, however, that the form of the customizable element in the compiled code on client 100 is in executable form since no source code is provided to client 100. Specifically, variable 700 is defined as a character type variable in, for example, C source code to have a default value 702 of "key_GOOD COMPANY". Default value 702 contains an embedded key 704 in the form of "key_". Variable 700 and its associated default value 702 are an example of a customizable element that can be searched for by tool 110 searching for key 704 and changed by the user to another value. When searching for each key as described above, editing tool 110 searches for the text string "key_". Variable 700 corresponds, for example, to the company name to be presented on a pop-up screen at the start-up of a browser program. If the user does not change this element, or the element is not changed because the user lacked authorization to make this change, then default value 702 is left unchanged as indicated by arrow 710. However, embedded key 704 has been replaced by customized flag 706.

If the user changes this element, then as indicated by arrow 712, default value 702 is replaced with the user selected value 708 of "USER COMPANY". Note that embedded key 704 is also replaced by customized flag 706. Thus, variable 700 can be modified by the user so that the customized compiled code assigns variable 700 with the initial value as customized by the user at run time. Other types of customized elements in the compiled code, such as those mentioned above, can be changed in a similar way.

In the preferred embodiment, variable 700, default value 702, and selected value 708, including embedded key 704 and customized flag 706, are incorporated into the compiled code in an encrypted form to prevent unauthorized modifications to the compiled code by the user or anyone else. This encryption is done to the original source code prior to compiling to generate the compiled code sent to client 100 and

may be done using conventional DES symmetric encryption or another standard encryption technique. The encrypted form of the variable and default value are inserted into the source code, which is then compiled. Tool 110 contains the unlocking encryption code necessary to decrypt values that it reads from the compiled code and to re-encrypt values to be inserted into the compiled code as user selected values. For example, embedded key 704 is stored in the compiled code in an encrypted form, but tool 110 searches for the encrypted form of the key in the compiled code. Note that to an unauthorized user encrypted key 704 will merely appear to be an unintelligible ASCII string.

Authorization of Customized Code

FIG. 5 is a flow diagram representing a preferred process for authorizing the compiled code received by acceptance unit 122 or server 102 from the user for authorization and unlocking. Customized code 500 has been uploaded from client 100. At step 502, customized code 500 is installed and tested for compliance to the desired requirements of the operator of server 102 or acceptance unit 122. In the preferred embodiment, customized code 500 includes the same self-extracting file as compiled and compressed at step 312 of FIG. 3.

In making the acceptance tests, the identification 504 of the user is considered to determine the level of authorization of the user and any requirements or agreements that the user should have complied with. Identification 504 may be for example the password discussed above and is sent by tool 110 as part of the compression and uploading process discussed above. As an example of this testing, a quality assurance (QA) person operating acceptance unit 122 receives the self-extracting file sent from client 100 and executes the file on acceptance unit 122 to install the customized code. The QA person then runs the program on acceptance unit 122 to observe and test its appearance and operation. In addition, the manifest list of customized values included

with the uploaded customized code 500 may be separately examined by the QA person and/or an automated software program running, for example, on acceptance unit 122 to quickly determine the content of any customized text or graphics added by the user.

At step 506, if the tested code is not acceptable, then at step 508 a rejection reply is sent to the user, for example, by E-mail. If the tested code is acceptable, then authorization program 120 running on either acceptance unit 122 or server 102 executes an unlocking authorization process on customized code 500.

Specifically, at step 510 authorization program 120 calculates a value for each customizable element in the customized code. In the preferred embodiment, authorization program 120 reads the customizable elements directly from the customized executable code sent by tool 110 from client 100--it is not necessary that these values be read from the separate manifest file. Each value is determined for a specific customizable element, for example, by examining each character in the text string for user selected value 708, or default value 702 where the user has not modified that element, and determining an integer value corresponding to the standard ASCII numerical code for that character. These ASCII codes are then summed for all characters in the element to determine a calculated value for that element. Each customizable element has a value calculated in this manner, whether or not that particular customizable element was modified by the user. The customizable elements may be located by searching for the encrypted form of customized flag 706 that has been previously inserted for each such element by tool 110. It should be noted that preferably all customizable elements will be used in the calculation to determine the authorization value.

After calculating a value for each element, at step 512 all such integer values are added together to get a total calculated authorization value for the entire customized code 500. At step 514 an unlock variable in the code 500 is searched and has its value updated directly in the executable form of the customized code 500 to the

authorization value to provide authorized code 516. Note that no source code is used and no recompilation is done here. In a preferred approach, the change in value of the unlock variable is the only change made to the code 500. The unlock variable and the stored authorization value are also preferably encrypted as discussed above for the customizable elements.

FIG. 8 illustrates an example of an unlock variable in source code form before and after authorization of the compiled code by the server computer. As mentioned above, uploaded customized code 500 is in compiled binary form and no source code modification or recompilation or linking is required as part of the unlocking authorization process. An unlock variable 800 is declared as a character type variable with an initial null value 802 (which corresponds to an unauthorized state). As indicated by arrow 804, unlock variable 800 is updated to store the calculated authorization value 806 determined above in step 512. The unlock variable 800 and authorization value 806 are stored as encrypted ASCII strings as discussed above for customizable elements.

By the use of unlock variable 800 and authorization value 806, improper modification of the compiled code after review and authorization is significantly hindered. Specifically, if any of the customizable elements are changed after authorization, the authorization value will no longer be valid. During execution, the application program will again determine a calculated value for all customizable elements and compare this value to the stored authorization value to determine if improper modifications have been made, as discussed in more detail below.

Running Authorized Customized Code on Client

FIG. 6 is a flow diagram representing a preferred process for client 100 to determine whether the compiled code has been altered after authorization by server 102 or acceptance unit 122. Specifically, downloaded authorized code 600, which

corresponds to download authorized code step 208 of FIG. 2, is received by client 100 from server 102, acceptance unit 122, or even optionally another computer in communication with server 102 or acceptance unit 122.

Authorized code 600 is preferably sent in a compressed self-extracting file created using authorized code 516 of FIG. 5. This self-extracting file can be created using the standard WISE compiler discussed above. At step 602, the self-extracting file is executed on client 100 to install and run authorized code 516.

As mentioned above, at step 604 the application program itself at run time calculates a value for each customizable element and sums these values in a similar manner as discussed above for authorization. At step 606, the calculated sum is compared to the previously-stored authorization value 806. If the sum and value are equal, then usage 608 is authorized and the program will no longer display the failed authorization message during operation. If the sum and value are not equal, then at step 610 the program will display an unauthorized usage message. For example, the sum and value will not be equal if the user has made a subsequent modification to a customizable element after receiving authorized code 600. This occurs because the unauthorized modified element will change the calculated value for the program, which will no longer equal the authorization value stored by the server operator.

Advantages

By the foregoing, a novel and unobvious method and system for customizing compiled code using an editing tool has been disclosed by way of preferred embodiments. By the use of the editing tool, a user, even one without software programming skill, can make substantial changes to the functionality of an application program by modifying its executable code without having access to the underlying source code and without going through a recompilation process.

The customization process above is also advantageous for the operation of a co-branding business with third parties in which the third party enters an agreement with the operator or entity in control of authorization of the customized code (the "authorizing party"). By such an agreement, the third party may be given different levels of authorization depending on the terms of the agreement and the distribution strategy of the authorizing party. Also, since the third party does not need access to the source code, the speed of entering agreements with potential distributors and authorizing code for distribution is greatly increased over traditional approaches. Further, since there is a high degree of control over the types of changes that can be made by such potential distributors, the quality of the customized product distributed is higher and the time required by the authorizing party and/or distributor to troubleshoot problems during customization is significantly reduced.

Also, other advantages are that the user does not require access to the installation source code or, for example, WISE compiler stored on the server computer in order to implement custom installations on the user's computer. Instead, the installation program provided to the client is pre-compiled, but configurable by use of the configuration file discussed above for FIG. 3 to mimic a custom-generated installation program. By providing a user with the pre-compiled customized browser installation program, it is not necessary to distribute the compiler itself to the user.

Other Variations

Although specific embodiments have been described above, numerous modifications and substitutions may be made without departing from the spirit of the invention. For example, while the description of preferred embodiments above sometimes discusses customization of an Internet-based browser, it is wholly within the purview of the invention customize software used for other applications such as,

for example, gaming software. Accordingly, the invention has been described by way of illustration rather than limitation.

5

CLAIMS

1. A method for customizing compiled code for an application program using an editing tool, comprising the steps of:

A. sending compiled code, comprising one or more embedded keys corresponding to one or more customizable elements of the application program, from a server computer to a client computer; and

B. sending an editing tool from the server computer to the client computer wherein the editing tool is operative to modify the compiled code by searching for the embedded keys to locate the customizable elements in the compiled code.

2. The method of claim 1 wherein all embedded keys contained within the compiled code are modified by the editing tool to be replaced by a customized flag to indicate that the compiled code has been modified.

3. The method of claim 2 wherein the editing tool comprises an embedded key searching program that will ignore the customized flag during scanning of the compiled code.

4. The method of claim 1 further comprising the steps of:

C. sending user identification information from the client computer to the server computer; and

D. sending an authorized list from the server computer of customizable elements that are authorized for customization on the client computer wherein the editing tool is operative to modify only the customizable elements on the authorized list.

5. The method of claim 4 wherein the user identification information corresponds to the client computer.

6. The method of claim 1 wherein:

each customizable element in the compiled code corresponds to a variable and an associated character string in a source code file used to generate the compiled code; and

wherein the variable and the associated character string are inserted into the source code in encrypted form prior to compiling the source code to generate the compiled code sent in step A.

7. The method of claim 6 wherein the variable and associated character string are encrypted using DES encryption.

8. The method of claim 2 wherein the customized flag is encrypted prior to insertion into the compiled code by the editing tool.

9. The method of claim 1 further comprising the steps of:

C. receiving from the client computer a copy of compiled code that has been modified using the editing tool; and

D. further modifying the compiled code then sending the modified compiled code to the client computer.

10. The method of claim 9 wherein the step of further modifying the compiled code comprises modifying a customizable item in the compiled code so that a calculated value corresponding to the customizable elements and a predetermined authorization value are equal.

11. The method of claim 10 wherein the application program displays a failed authorization message to a user if the predetermined authorization value and the calculated value are not equal.

12. The method of claim 11 wherein the failed authorization message is a pop-up window containing a message that use of the application program is not authorized.

13. The method of claim 10 wherein the calculated value is determined by a procedure comprising the steps of:

E. determining a text string value corresponding to each customizable element in the compiled code; and

F. adding together each text string value determined in step E to determine the calculated value.

14. The method of claim 1 wherein the customizable elements comprise one or more elements corresponding to the appearance and functionality of the application program.

15. The method of claim 14 wherein the elements comprise a network file location designation of a file accessed by the application program during operation.

16. The method of claim 15 wherein the file location designation is a uniform resource locator.

17. The method of claim 1 wherein the editing tool is further operative to communicate with the server computer prior to modifying the compiled code.

18. The method of claim 17 wherein the editing tool is further operative to communicate with the server computer prior to modifying the compiled code by a procedure comprising the steps of:

C. sending an identification code to the server computer corresponding to a user of the application program; and

D. receiving a list from the server computer corresponding to selected customizable elements to be modified by the editing tool.

19. The method of claim 18 wherein the application program corresponding to the compiled code is a program to provide a user interface for interacting with a multiple computer network.

20. The method of claim 19 wherein the multiple computer network is a public computer network, the Internet, or a private intranet.

21. The method of claim 1 wherein the embedded keys are encrypted prior to integrating into the compiled program.

22. The method of claim 21 wherein the embedded keys correspond to a common encrypted ASCII string integrated into the compiled code.

23. A system for customizing compiled code for an application program using an editing tool comprising:

a server computer;

a client computer operative to communicate with the server computer;

compiled code, comprising one or more embedded keys corresponding to one or more customizable features of the application program, residing on the client computer; and

an editing tool, operative to modify the compiled code by searching for the embedded keys to locate the customizable elements in the compiled code, residing on the client computer.

24. The system of claim 23 further comprising an authorization program stored on the server computer operative to further modify the compiled code after modification by the editing tool.

25. A method for modifying compiled code comprising one or more customizable elements, comprising the steps of:

A. downloading the compiled code to a client computer;

B. sending a user identification, corresponding to the client computer, from the client computer to a server computer;

C. sending an authorized list, corresponding to the user identification, of the customizable elements that are authorized for modification to the client computer; and

D. modifying one or more of the customizable elements on the authorized list to generate customized code.

26. The method of claim 25 further comprising the steps of:

E. uploading the customized code generated in step D to an acceptance unit;

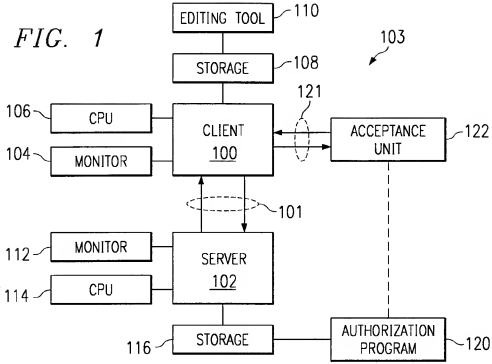
F. modifying the customized code uploaded in step E to generate authorized code; and

G. downloading the authorized code to the client computer.

27. The method of claim 26 wherein the customized code uploaded in step E comprises a manifest file created by the editing tool for providing a separate list of the custom values selected by the user for the customizable elements modified by the user in step D.

28. The method of claim 26 wherein the acceptance unit is the server computer.

FIG. 1


EDITING TOOL
110
103
STORAGE
108
121
106
CPU
CLIENT
100
ACCEPTANCE
UNIT
122
104
MONITOR
101
112
MONITOR
SERVER
102
114
CPU
116
STORAGE
AUTHORIZATION
PROGRAM
120

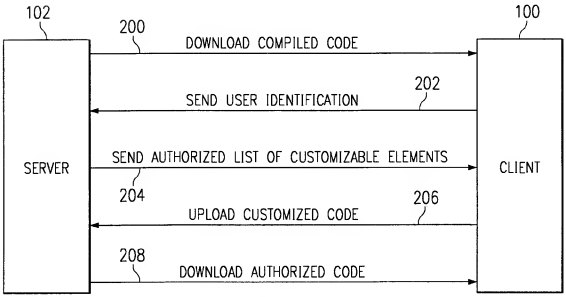

102
100
200
DOWNLOAD COMPILED CODE
202
SEND USER IDENTIFICATION
SERVER
SEND AUTHORIZED LIST OF CUSTOMIZABLE ELEMENTS
CLIENT
204
206
UPLOAD CUSTOMIZED CODE
208
DOWNLOAD AUTHORIZED CODE

FIG. 2

FIG. 3

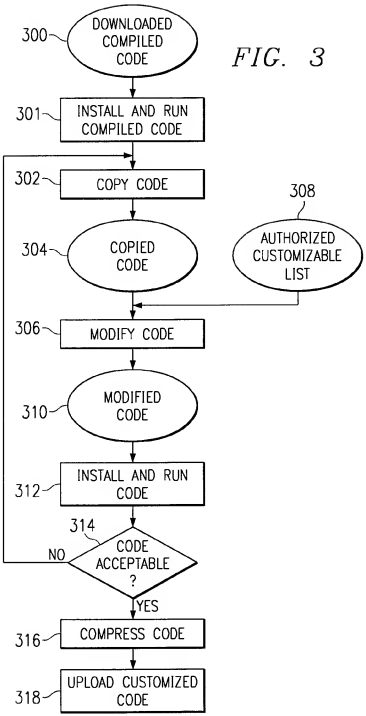
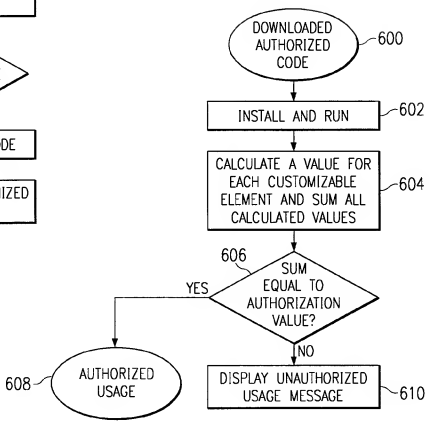

300
DOWNLOADED COMPILED CODE
301
INSTALL AND RUN COMPILED CODE
302
COPY CODE
304
COPIED CODE
308
AUTHORIZED CUSTOMIZABLE LIST
306
MODIFY CODE
310
MODIFIED CODE
312
INSTALL AND RUN CODE
314
CODE ACCEPTABLE ?
NO
YES
316
COMPRESS CODE
318
UPLOAD CUSTOMIZED CODE

FIG. 6


DOWNLOADED AUTHORIZED CODE
600
INSTALL AND RUN
602
CALCULATE A VALUE FOR EACH CUSTOMIZABLE ELEMENT AND SUM ALL CALCULATED VALUES
604
606
SUM EQUAL TO AUTHORIZATION VALUE?
YES
NO
608
AUTHORIZED USAGE
DISPLAY UNAUTHORIZED USAGE MESSAGE
610

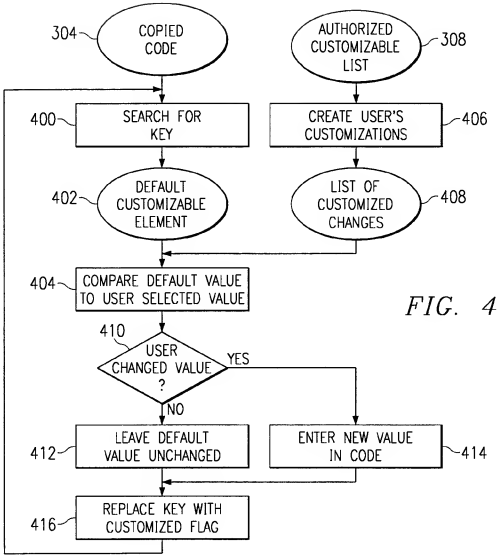

304
COPIED CODE
AUTHORIZED CUSTOMIZABLE LIST
308
400
SEARCH FOR KEY
CREATE USER'S CUSTOMIZATIONS
406
402
DEFAULT CUSTOMIZABLE ELEMENT
LIST OF CUSTOMIZED CHANGES
408
404
COMPARE DEFAULT VALUE TO USER SELECTED VALUE
410
USER CHANGED VALUE ?
YES
NO
412
LEAVE DEFAULT VALUE UNCHANGED
ENTER NEW VALUE IN CODE
414
416
REPLACE KEY WITH CUSTOMIZED FLAG

FIG. 4

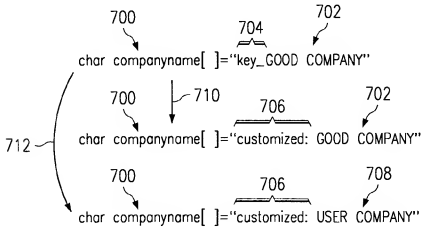

700
704
702
char companyname[ ]="key_GOOD COMPANY"
700
710
706
702
712
char companyname[ ]="customized: GOOD COMPANY"
700
706
708
char companyname[ ]="customized: USER COMPANY"

FIG. 7

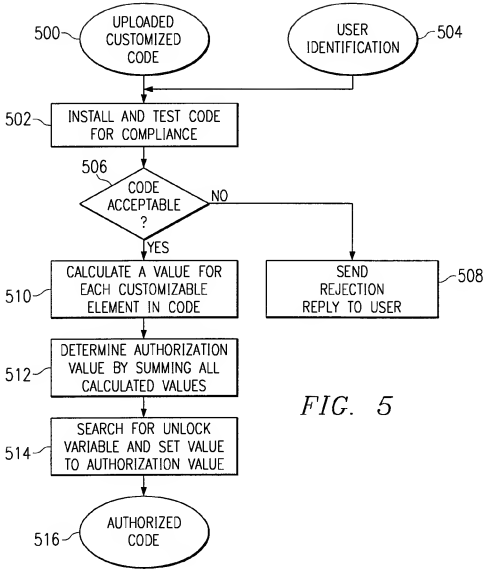

500
UPLOADED CUSTOMIZED CODE
USER IDENTIFICATION
504
502
INSTALL AND TEST CODE FOR COMPLIANCE
506
CODE ACCEPTABLE ?
NO
YES
510
CALCULATE A VALUE FOR EACH CUSTOMIZABLE ELEMENT IN CODE
SEND REJECTION REPLY TO USER
508
512
DETERMINE AUTHORIZATION VALUE BY SUMMING ALL CALCULATED VALUES
514
SEARCH FOR UNLOCK VARIABLE AND SET VALUE TO AUTHORIZATION VALUE
516
AUTHORIZED CODE

FIG. 5

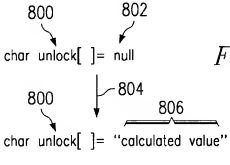

800
802
char unlock[ ]= null
804
800
806
char unlock[ ]= "calculated value"

FIG. 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/22638

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) :G06F 9/445
US CL : 717/5
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 717/1, 2, 4, 5, 11

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

MICROSOFT MSDN LIBRARY

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

WEST, ACM, IEEE, PROQUEST
search terms: customize, browser, compiled code, tools, flag, keys, download

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X ---- Y	M2 COMMUNICATIONS LTD., NETSCAPE: Free Client Customization Kit for ISPS, OEMS and Internet Content Providers Worldwide, M2 PRESSWIRE, March 1998, pp. 1-3, entire document.	1, 4-7, 9, 14-22, 23-24, 25-28 ---------- 2-3, 8, 10-13
X ---- Y	M2 COMMUNICATIONS LTD., MICROSOFT: Leading Internet Content Providers Choose Microsoft Internet Explorer 5, M2 PRESSWIRE, May 1999, pp.1-3, entire document.	1, 4-7, 9, 14-22, 23-24, 25-28 ---------- 2-3, 8, 10-13

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search	Date of mailing of the international search report
19 OCTOBER 2000	24 NOV 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer Mark R Powell James R. Matthews Telephone No. (703) 308-9049

Form PCT/ISA/210 (second sheet) (July 1998)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/22638

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X,P ---- Y,P	M2 COMMUNICATIONS LTD., NOKIA: Nokia to license WAP 1.2 Compliant Browser as Source Code; Nokia WAP Browser Enables Customers to Offer Customized User Interface and Integrate Value Added Functionality, M2 PRESSWIRE, August 2000, pp. 1-2, see entire document.	1, 4-7, 9, 14-22, 23-24, 25-28 ---------- 2-3, 8, 10-13
Y	DEVANBU et al., CHIME: Customizable Hyperlink Insertion and Maintenance Engine for Software Engineering Environments, ACM, May 1999, pp.473-482, see entire document.	2-3, 8, 10-13
Y	CHAKRABARTI et al., IMIS A Tool for Developing MIS for Internet/Intranet Environment, IEEE, December 1998, pp. 17-20, see entire document.	2-3, 8, 10-13
A	ESPOSITO, Browser Helper Objects: The Browser The Way You Want It, MICROSOFT CO., http://msdn.microsoft.com/library/techart/bho.htm , January 1999, pp. 1-21, entire document.	1-28
A	ROBERTS, Take Total Control of Internet Explorer with Advanced Hosting Interface, MICROSOFT CO., http://msdn.microsoft.com/library/periodic/period98/advhost.ht m , October 1998, pp. 1-11, entire document.	1-28
A	HENRY, Customizing The Web, COMPUTER RESELLER NEWS, March 1998, pp. 1-4, entire document.	1-28
A	GALLAGHER, Browsers For E-Mail, INFORMATIONWEEK LABS, May 1997, pp. 63-64, 66, 72, 76, entire document.	1-28
A	CHAMBERS et al., Customization: Optimizing Compiler Technology for SELF, a Dynamically-Typed Object-Oriented Programming Language, ACM, June 1989, pp. 146-160, entire document.	1-28
A	MCDONALD, McAfee Debuts Customized Browser, PC WORLD ONLINE, March 1999, pp. 1-2, entire reference.	1-28
A	US 5,644,334 A (JONES et al.) 01 July 1997, entire reference.	1-28

DERWENT-ACC-NO: 2001-441066

DERWENT-WEEK: 200147

TITLE: Compiled code customizing method involves sending editing tool which modifies compiled code by searching embedded keys to locate customizable elements, from server to client

INVENTOR: COHEN D; COX B G ; FRISKEL J ; MARTINEAU C P ; SIELAFF M E

PATENT-ASSIGNEE: NEOPLANET INC[NEOPN]

PRIORITY-DATA: 1999US-385779 (August 30, 1999)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE
WO 0116731 A1	March 8, 2001	EN
AU 200069140 A	March 26, 2001	EN

DESIGNATED-STATES: AE AL AM AT AU AZ BA BB BG BR BY CA CH CN CR CU CZ DE DK DM EE ES FI GB GD GE GH GM HR HU ID IL IN IS JP KE KG KP KZ LC LK LR LS LT LU LV MA MD MG MK MN MW MX NO NZ PL PT RO RU SD SE SG SI SK SL TJ TM TR TT TZ U A UG US UZ VN YU ZA ZW AT BE CH CY DE DK EA ES FI FR GB GH GM GR IE IT KE LS LU MC MW MZ NL OA PT SD SE SL SZ TZ UG ZW

APPLICATION-DATA:

PUB-NO	APPL-DESCRIPTOR	APPL-NO	APPL-DATE
WO2001016731A1	N/A	2000WO-US22638	August 17, 2000
AU 200069140A	Based on	2000AU-069140	August 17, 2000

INT-CL-CURRENT:

TYPE	IPC DATE
CIPS	G06F9/445 20060101

ABSTRACTED-PUB-NO: WO 0116731 A1

BASIC-ABSTRACT:

NOVELTY - The compiled code with embedded keys to locate the customizable elements of application program, is sent from the server (102) to the client (100). The editing tool which modifies the compiled code by searching for the embedded keys is sent from the server to the client.

DESCRIPTION - An INDEPENDENT CLAIM is also included for compiled code customizing system.

USE - For customizing the compiled code of an application program such as Internet browser.

ADVANTAGE - An user without software programming

skill can modify the compiled code, without having access to underlying source code and without going through a recompilation process. Since modifications are done only in the executable code, the sources code having secrets is maintained confidently. The user does not need any information from the source code and the recompilation is not needed. Since there is a high degree of control over the types of modifications, the quality of the customized code is higher and the time required for customizing is reduced.

DESCRIPTION OF DRAWING(S) - The figure shows the client-server data communications.

Client (100)

Server (102)

CHOSEN-DRAWING: Dwg.2/8

TITLE-TERMS: COMPILE CODE METHOD SEND EDIT TOOL MODIFIED SEARCH EMBED KEY LOCATE ELEMENT SERVE CLIENT

DERWENT-CLASS: T01

EPI-CODES: T01-F05A; T01-F05G5; T01-H07C5S;

SECONDARY-ACC-NO:

Non-CPI Secondary Accession Numbers: 2001-326302